Processamento da Informação
– Teoria –

Funções

Semana 01

Prof. Jesús P. Mena-Chalco

27/04/2013

Funções

No contexto de linguagens de programação, uma **função** é uma sequência de instruções utilizada para realizar alguma operação.

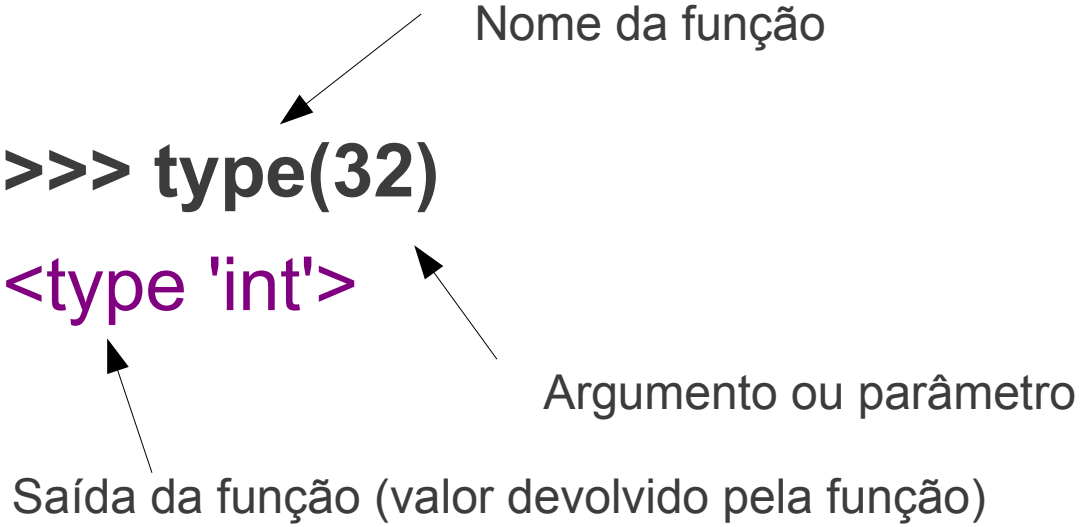
Uma função pode ser executada mais de uma vez em um programa.

```
>>> type(32)
<type 'int'>
```

Funções

No contexto de linguagens de programação, uma **função** é uma sequência de instruções utilizada para realizar alguma operação.

Uma função pode ser executada mais de uma vez em um programa.



Funções

No contexto de linguagens de programação, uma **função** é uma sequência de instruções utilizada para realizar alguma operação.

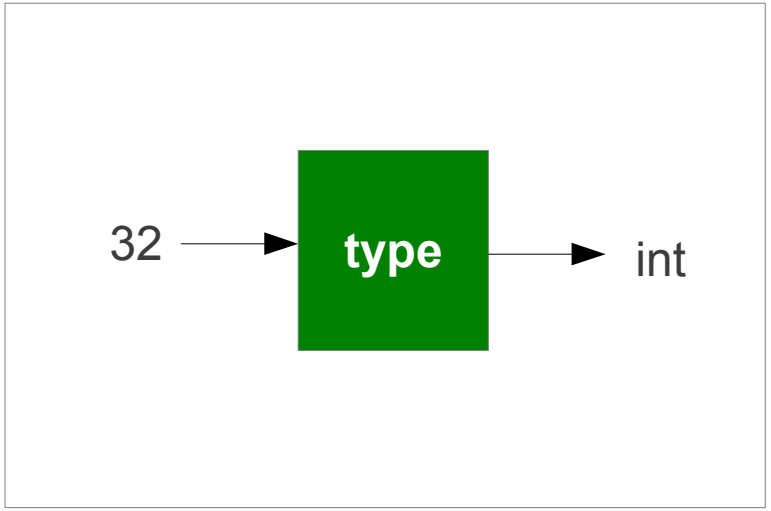
Uma função pode ser executada mais de uma vez em um programa.

Nome da função

```
>>> type(32)
<type 'int'>
```

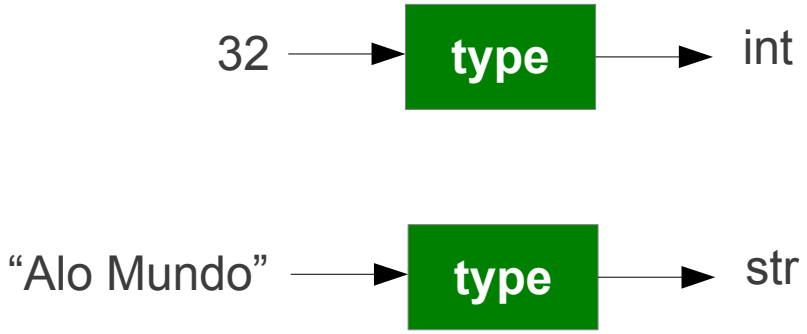
Argumento ou parâmetro

Saída da função (valor devolvido pela função)

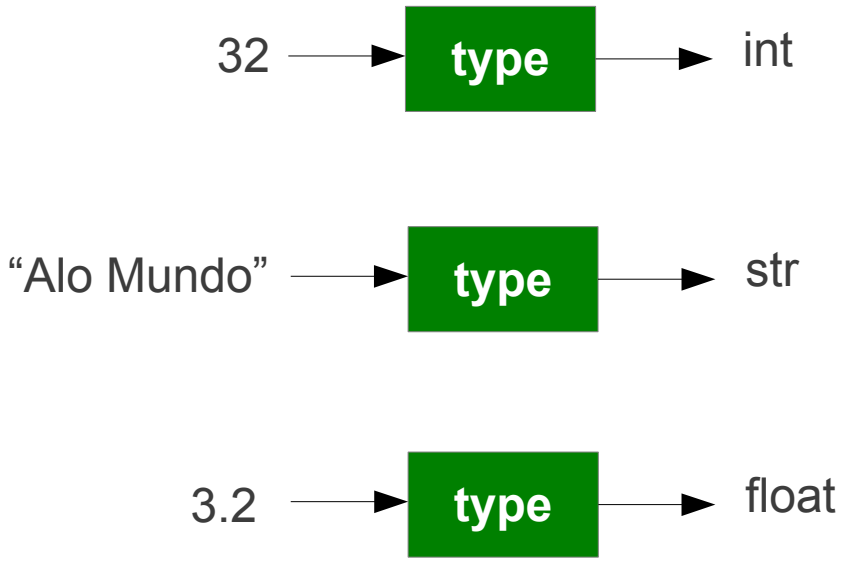


Funções pré-definidas na linguagem de programação

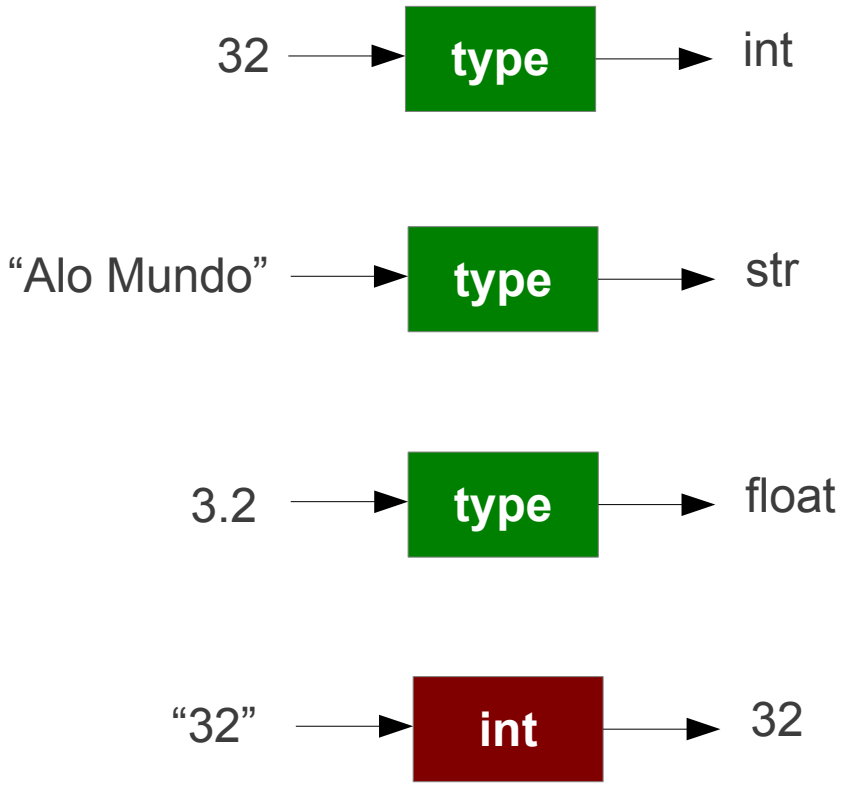
Funções pré-definidas na linguagem de programação



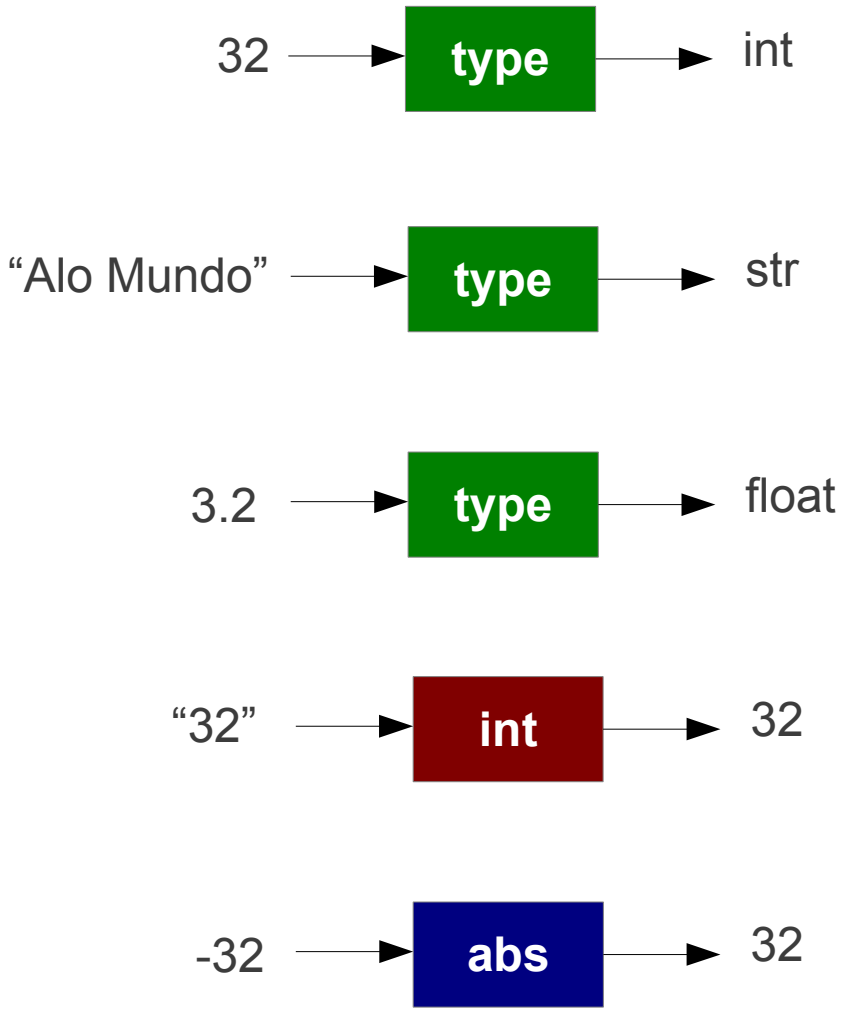
Funções pré-definidas na linguagem de programação



Funções pré-definidas na linguagem de programação



Funções pré-definidas na linguagem de programação

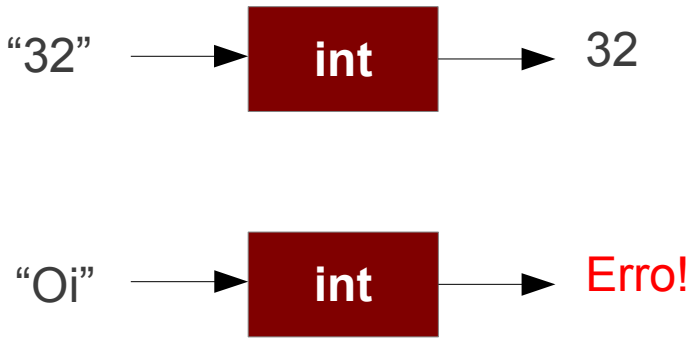


Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).

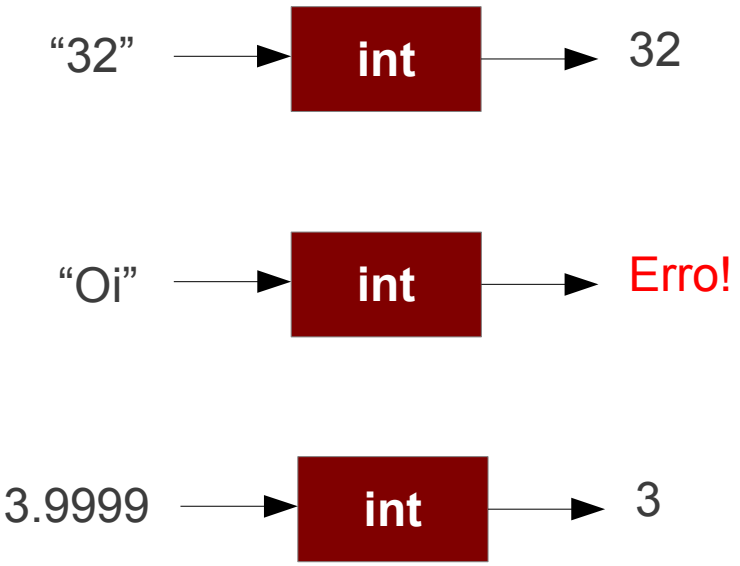
Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).



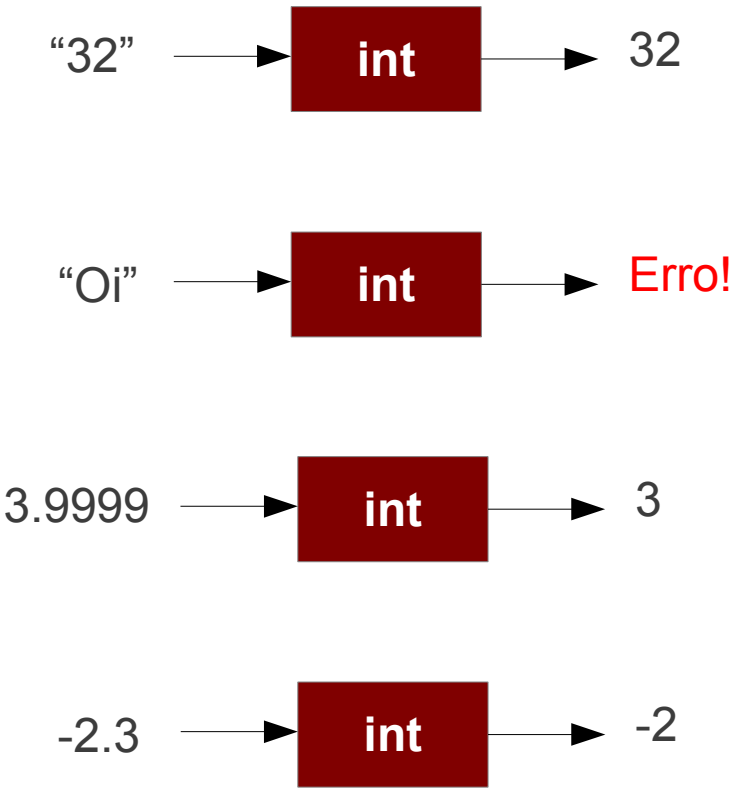
Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).



Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).

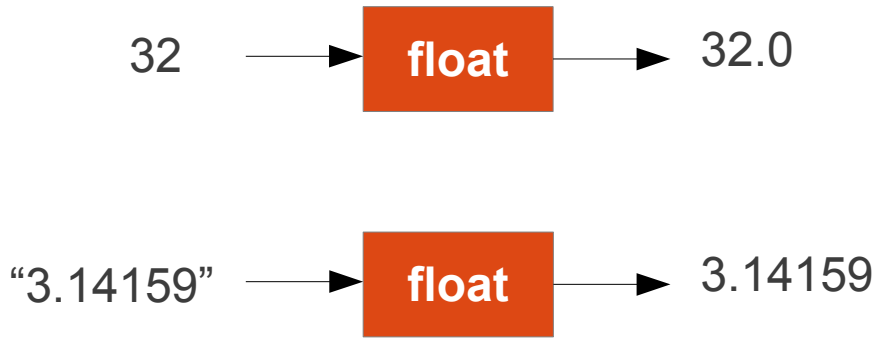


Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).

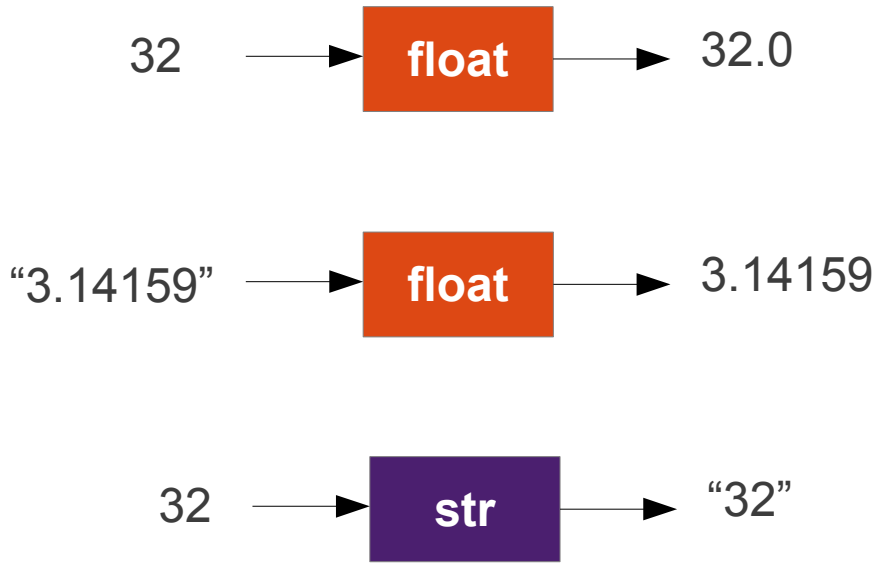
Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).



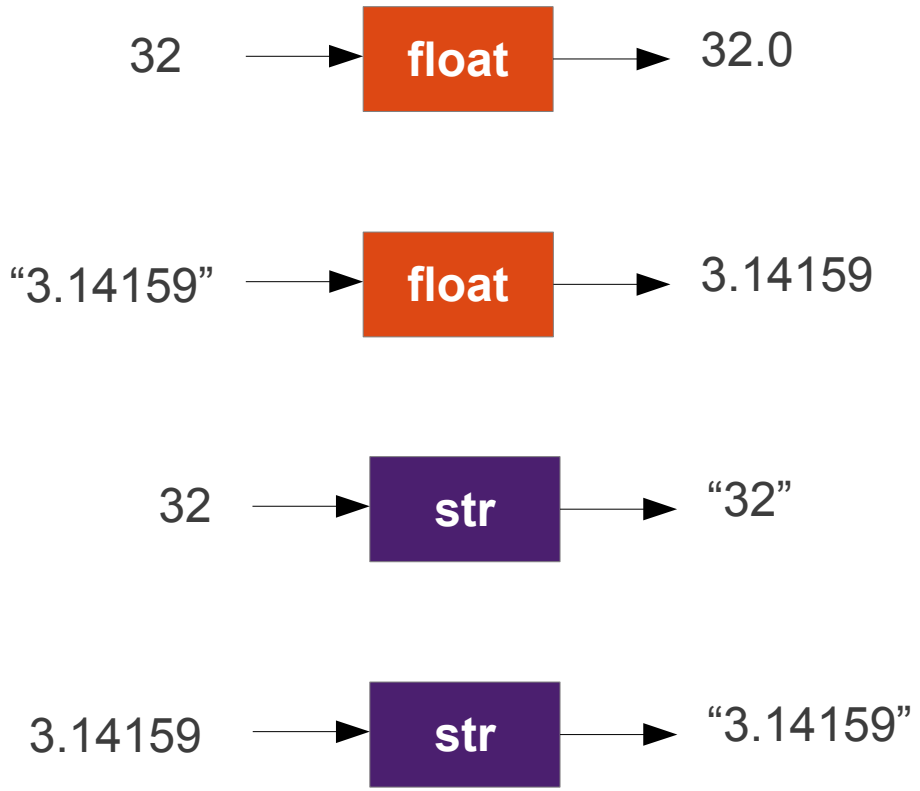
Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).



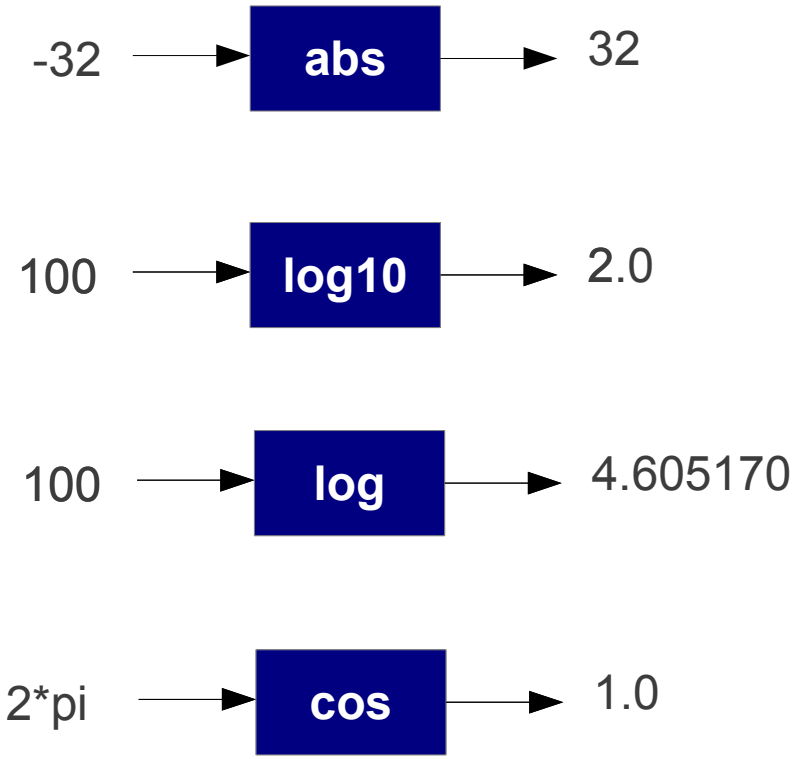
Funções de conversão de tipo

Essa classe de funções permite converter o tipo de dado de um elemento (dado como parâmetro).



Funções matemáticas

Essa classe de funções permite trabalhar com algumas operações matemáticas.



Outras funções: sqrt, sin, tan, exp (e=2.718281...)

Composição de operações

Nos parâmetros das funções também podem ser utilizadas composições entre: variáveis e expressões.

```
>>> x = sin(2*pi+4)

>>> y = exp(log(x+1))
```

Composição de operações

Nos parâmetros das funções também podem ser utilizadas composições entre: variáveis e expressões.

```
>>> horas = 2

>>> minutos = horas * 60

>>> horas * 60 = minutos
SyntaxError: can't assign to operator
```

Funções de usuário

A maioria das linguagens de programação nos permite escrever nossas próprias funções.

Ao escrever nossas próprias funções podemos “quebrar” um programa complexo em vários subprogramas.

Depois voltaremos a tratar algumas das vantagens da criação de funções (modularizar):

- Reutilização de instruções.
- Decomposição procedural.

Funções de usuário: exemplo

```
def imprime_apresentacao():
    print "Bem-vindo à disciplina"
    print "Processamento da Informação"
    print "UFABC"


def imprime_sala():
    print "Sala A-103-0"
```

Funções de usuário: exemplo

```
def imprime_apresentacao():
    print "Bem-vindo à disciplina"
    print "Processamento da Informação"
    print "UFABC"
```

Cabeçalho da função

Corpo da função

Vamos adotar, na disciplina, uma indentação de **4** caracteres.

Funções de usuário: exemplo

```
def somar_numeros(x, y):
    z = x+ y
    return z
```

Funções de usuário: exemplo

```
def somar_numeros(x, y):
    z = x+ y
    return z
```

```
def somar_numeros(x, y):
    return x+y
```

Funções de usuário: exemplo

```
def calcular_nota(p1, p2, listas):
    return 0.3*p1 + 0.4*p2 + 0.3*listas


def determinar_o_maior_numero(p, q):
    if p<q:
        return q
    else:
        return p
```

Funções de usuário: exemplo

```
def potencia(base, expoente):
    resposta = base**expoente
    return resposta


def operacoes_magicas(n1, n2, n3):
    operacao1 = potencia(n1, n2)
    operacao2 = potencia(operacao1, n3)
     return operacao2
```

Funções de usuário: exemplo

```
def operacoes_magicas(n1, n2, n3):
    operacao1 = potencia(n1, n2)
    operacao2 = potencia(operacao1, n3)
    return operacao2

def operacoes_magicas(n1, n2, n3):
    return potencia(potencia(n1, n2), n3)

>>> operacoes_magicas(2,3,2)
```

Funções de usuário: exemplo

```
def operacoes_magicas(n1, n2, n3):
    operacao1 = potencia(n1, n2)
    operacao2 = potencia(operacao1, n3)
    return operacao2

def operacoes_magicas(n1, n2, n3):
    return potencia(potencia(n1, n2), n3)

>>> operacoes_magicas(2,3,2)
    64
```

Funções de usuário: exemplo

```
def operacoes_magicas(n1, n2, n3):
    operacao1 = potencia(n1, n2)
    operacao2 = potencia(operacao1, n3)
    return operacao2

def operacoes_magicas(n1, n2, n3):
    return potencia(potencia(n1, n2), n3)

>>> operacoes_magicas(2,3,2)**(0.5)
```

Funções de usuário: exemplo

```
def operacoes_magicas(n1, n2, n3):
    operacao1 = potencia(n1, n2)
    operacao2 = potencia(operacao1, n3)
     return operacao2

def operacoes_magicas(n1, n2, n3):
    return potencia(potencia(n1, n2), n3)

>>> operacoes_magicas(2,3,2)**(0.5)
      8
```

Atividade em aula 01

Dadas as seguintes funções:

```
def equacao1(p, q):
    r1 = p+q
    r2 = p-q
    return r1*r2

def equacao2(r, s):
    return r**2 – s**2
```

Determine os valores para as seguintes operações:

(1) equacao1(3,4)

(2) equacao1(4,3)

(3) 2**equacao2(2,0)

(4) equacao1(0,2) + equacao2(0,4)

(5) equacao1(9,99)-equacao2(9,99)

Atividade em aula 01

Dadas as seguintes funções:

```
def equacao1(p, q):
    r1 = p+q
    r2 = p-q
    return r1*r2

def equacao2(r, s):
    return r**2 – s**2
```

Determine os valores para as seguintes operações:

(1) equacao1(3,4) → -7

(2) equacao1(4,3) → 7

(3) 2**equacao2(2,0) → 16

(4) equacao1(0,2) + equacao2(0,4) → -20

(5) equacao1(9,99)-equacao2(9,99) → 0

Atividade em aula 02

Dadas as seguintes funções:

```
def eh_numero_par(x):
    if x%2==0:
        return True
    else:
        return False

def funcaoX(a, b):
    if eh_numero_par(a):
        return a-b
    else:
        return a-2*b
```

Determine os valores para as seguintes operações:

(1) funcaoX(0,20)

(2) funcaoX(20,3)

(3) funcaoX(3,20)

(4) eh_numero_par(1)+eh_numero_par(2)

(5) eh_numero_par(4)*funcaoX(1, funcaoX(2,3))

Atividade em aula 02

Dadas as seguintes funções:

```
def eh_numero_par(x):
    if x%2==0:
        return True
    else:
        return False

def funcaoX(a, b):
    if eh_numero_par(a):
        return a-b
    else:
        return a-2*b
```

Determine os valores para as seguintes operações:

(1) funcaoX(0,20) → -20

(2) funcaoX(20,3) → 17

(3) funcaoX(3,20) → -37

(4) eh_numero_par(1)+eh_numero_par(2)
→ 1

(5) eh_numero_par(4)*funcaoX(1, funcaoX(2,3))
→ 3

Qual a vantagem de usar funções?

- Podem se dividir um problema em subproblemas
- Funções menores facilitam o entendimento da solução.
- Pode se dividir o desenvolvimento entre vários programadores.
- Reutilização de instruções.

Lista 01

Exercício 1:

Crie uma função que permita a conversão de graus celsius para fahrenheit. O cabeçalho da função deve ser o seguinte:

```
def celcius2fahrenheit(graus):
```

A entrega da Lista 01 deverá ser realizada através do Tidia-ae. Seção Atividades/Lista-01. Até 04/05 (23h50).

Apenas deve ser enviado um arquivo PDF contendo a solução do exercício. O documento deve ter o seguinte nome: RA-SeuNomeCompleto-Lista-01.pdf